Reducing Software Projects Duration Using C#

Marius VETRICI
Economic Informatics Department - Academy of Economic Studies
mariusvetrici@softmentor.ro

This paper analyzes the impact of using the C# language on project development duration. A set of duration estimation techniques are explained, the difficulties of estimating software project duration are enumerated and the factors that negatively impact the duration of a software project are analyzed. The technical advantages of C# language are presented as factors that reduce project complexity and duration.

JEL classification: l86 computer software, o22 project analysis

Keywords: Project duration, estimation models, influence factors.

Introduction
Several studies made over the past ten years revealed the dramatic reality of the software projects: one out of the four software projects is considered successful; the effective projects duration is 222% over the schedule and the resources are 189% higher then budgeted [1], [2], [3], [4]. These staggering numbers put actual and potential investors into great difficulties because these results translate into great capital losses and high risks for further investments into software.

The Standish Group’s survey gives us a useful insight into the factors that mostly challenge a software project [4]:

<table>
<thead>
<tr>
<th>No.</th>
<th>Project Challenge Factors</th>
<th>% of responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lack of User Input</td>
<td>12.8%</td>
</tr>
<tr>
<td>2</td>
<td>Incomplete Requirements &amp; Specifications</td>
<td>12.3%</td>
</tr>
<tr>
<td>3</td>
<td>Changing Requirements &amp; Specifications</td>
<td>11.8%</td>
</tr>
<tr>
<td>4</td>
<td>Lack of Executive Support</td>
<td>7.5%</td>
</tr>
<tr>
<td>5</td>
<td>Technology Incompetence</td>
<td>7.0%</td>
</tr>
<tr>
<td>6</td>
<td>Lack of Resources</td>
<td>6.4%</td>
</tr>
<tr>
<td>7</td>
<td>Unrealistic Expectations</td>
<td>5.9%</td>
</tr>
<tr>
<td>8</td>
<td>Unclear Objectives</td>
<td>5.3%</td>
</tr>
<tr>
<td>9</td>
<td>Unrealistic Time Frames</td>
<td>4.3%</td>
</tr>
<tr>
<td>10</td>
<td>New Technology</td>
<td>3.7%</td>
</tr>
<tr>
<td></td>
<td>Other</td>
<td>23.0%</td>
</tr>
</tbody>
</table>

This paper addresses the fifth challenge factor “Technology Incompetence” as a possible cause for project delay. The more complex a software development technology, the greater the technology incompetence and the fewer skillful software developers are using that technology. The reverse is also true: the easier a technology is (considering a constant productivity lever), the greater technology competence and the fewer challenged projects.

2. Project duration estimation
2.1 Definitions
A project is “a temporary endeavor undertaken to create a unique product, service, or result” [5]. A software development project is a temporary endeavour undertaken to create a unique piece of software. High quality software development projects deliver the required product within scope, on time and within budget. It is the project manager’s duty to skillfully balance the competing demands for project quality, project duration and cost of resources in order to be able to deliver the software as planned.

Like any other type of project, software development projects need:
- clearly defined requirements and scope
- established achievable objectives
- controlled resource allocation
- good effort and schedule management

The expectations of stakeholders are focused on the software to be delivered, on the budgeted consumption and on the project duration.
The duration of a project is the time elapsed between the project start and the project delivery date, when the software is delivered to the customer. The project duration is an essential indicator that should be well estimated, agreed upon with the stakeholders and thoroughly monitored, up to the project completion.

2.2 The importance of estimating project duration

Project duration and size reflect the manager’s own understanding of the requirements. It is not possible to correctly size and estimate duration for a project that is not completely understood. Further, project duration provides an important check for scope creep throughout the project. Failing to pay attention to project duration one could agree to add new functionality without appropriately updating project size and effort needed.

2.3 The difficulties of estimating software project duration

There are several reasons that make software project duration estimation a difficult problem. First of all, the very essence of software building process makes it difficult to measure. It is a tough endeavor to try to measure “how much” software is there in a software project because the software is invisible and unvisualizable [6]. This especially difficult if we try to make such forecasts before a detailed software design.

The software is pure thought-stuff, infinitely malleable [6]. Unlike cars and buildings, the software is constantly subject to pressures for change because the costs of modifications are difficult to understand.

Many of the classic problems of developing software products derive from this essential complexity and its nonlinear increases with size. From the complexity comes the difficulty of communication among team members, which leads to product flaws, cost overruns and schedule delays. From the complexity comes the difficulty of enumerating, much less understanding, all the possible states of the program, and from that comes the unreliability [7].

2.4 Duration estimation techniques

The grand majority of techniques for project duration estimation can be found either in bottom-up or top-down category. The difference between the two comes from the approach used to estimate project duration. The techniques in the first category start at the task-level view of the project and aggregate the work to be performed on higher levels, up to the project as a whole. The top-down way offers duration predictions based on properties of the work-product, the project team, and the project environment, figure 1.

Fig.1. Bottom-up vs. top-down techniques

*The bottom-up techniques* start with developing a work breakdown structure of the work and then continue with task identification and task duration estimation. Every task should be simple enough so as one could easily answer the question regarding the task duration three parameter estimates:

- best duration estimation
- most likely
- worst duration

Also for every task one should know:
- what's involved in getting started
- how will resources be allocated
- what exactly are the conditions to be met in order the project to be considered done.

The next step is identifying the predecessor-successor relationships and the critical path through the activity graph. In order to forecast the completion time, three different approaches can be used.

a) The simple approach consists in adding-up the most likely estimates for each task on the critical path. It is not the best method, but it is the simplest one.

b) The second approach means to calculate the expected task duration $ED$ as a weighted mean of the three given estimations using PERT equation:

$$ED = \frac{BD + 4 \times MD + WD}{6}$$  \hspace{1cm} (1)
where:
BD – best duration estimation; this is the most optimistic expectation, the best case scenario that assumes no influence is going to negatively impact the project duration;
MD – most likely; the duration of activity given the resources, their productivity and realistic expectations of availability;
WD – worst duration; the duration of activity based on a worse case scenario of what is described in most likely estimate.
c) The third approach relies on a Monte Carlo simulation over the task estimation data. The result will be a probabilistic distribution of the project duration.

The top-down techniques use instead some high level attributes of the project (related to its complexity, functionality or size) and of the organization capability to deliver the project. Top-down estimation begins with an assessment of the size of the work-product being planned. This idea comes from the construction projects, where the project-manager wouldn't imagine committing to a deadline without establishing and tracking some good size estimates, like the number of square feet, number of windows, doors, etc. to be designed and built.

Up to date there are four software project sizing legacy methods. See table 1 [11]:

<table>
<thead>
<tr>
<th>Sizing Method</th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of Codes</td>
<td>Easy to measure in many development environments (after there is code).</td>
<td>Cannot be done before there are lines of code.</td>
</tr>
<tr>
<td>Function Points</td>
<td>Can be measured during requirements stage.</td>
<td>Requires some training, calibration and perhaps tailoring to specific application domains.</td>
</tr>
<tr>
<td>Use-Case Counting</td>
<td>Can be measured during requirements</td>
<td>New method. Small experience base at this time.</td>
</tr>
</tbody>
</table>

The next step in top-down estimation is to use a project duration estimation model. Lawrence Putnam proposed a widely used model for project duration estimation using data on size, effort, and historic duration for thousands of other software projects. The model builds up the organization’s delivery capability index using PP - Productivity Parameter and links it to size, effort and duration dynamics.

\[
PP = \frac{PS}{\left(\frac{E}{\beta}\right)^{1/3} D^{4/3}} \tag{2}
\]

where:
PP – Putnam’s productivity index. This item shows the organization’s project delivery capability;
PS – project size, counted using one of the above sizing methods;
E – effort (in man-years). The work needed in order to fulfill the project;
D – the project duration (years).

The following things are notable in regard to this model:
a) an organization with higher PP can deliver more size with less effort and in shorter duration than one with a lower PP;
b) the 1/3 and 4/3 exponents in equation 2 express the non-linearity in effort-duration relationship.

2.5 Choosing a project duration estimation technique

Both top-down and bottom-up approaches proved to be good at estimating project duration. A good software project manager will probably use both methods, plus his own estimation, based on priori experience. Bottom-up estimates use work-breakdown structure, critical path method and task estimates; they provide crucial details regarding the duration
of smaller project parts and they rollup to a
global duration and effort estimation. Top-
down estimates rely on history of other real
projects. One's cumulative experience in
similar projects can provide estimates that
deserve some consideration in balance with
the bottom-up and top-down views.

3. C# language for software projects
3.1 Language overview
C# is an elegant and type-safe object-
oriented language that enables developers to
build a wide range of secure and robust ap-
lications that run on the .NET Framework.
C# can be successfully used to create tradi-
tional Windows client applications, XML
Web services, distributed components, client-
server applications, database applications,
and much, much more [13], [14].
C# programs run on the .NET Framework, an
integral component of Windows that includes
a virtual execution system called the Com-
mon Language Runtime - CLR and a unified
set of class libraries. The CLR is Microsoft's
commercial implementation of the Common
Language Infrastructure - CLI, an interna-
tional standard that is the basis for creating
execution and development environments in
which languages and libraries work together
seamlessly. See figure 1.
In addition to these basic object-oriented
principles, C# facilitates the development of
software components through several innova-
tive language constructs, including:
• encapsulated method signatures called
deblegates, which enable type-safe event noti-
fications.
• properties, which serve as accessors for
private member variables.
• attributes, which provide declarative meta-
data about types at run time.
• inline XML documentation comments.

3.2 C# language evolution
C# language evolved from the success and
failures of earlier languages like C, C++ and
Java. Developers who know any of these
languages are typically able to begin working
productively in C# within a very short time.
C# syntax simplifies many of the complexi-
ties of C++ while providing powerful fea-
tures such as nullable value types, enumera-
tions, delegates, anonymous methods and di-
rect memory access, which are not found in
Java. C# also supports generic methods and
types, which provide increased type safety
and performance, and iterators, which enable
implementers of collection classes to define
custom iteration behaviors that are simple to
use by client code [6], [8].

3.3 C# language advantages for reducing pro-
ject durations
The C# build process is simple compared to
C and C++ and more flexible than in Java.
There are no separate header files, and no re-
quirement that methods and types be de-
clared in a particular order. A C# source file
define any number of classes, structs, in-
terfaces, and events.
Language interoperability is a key feature of
the .NET Framework. Because the IL code
produced by the C# compiler conforms to the
Common Type Specification - CTS, IL code
generated from C# can interact with code that
was generated from the .NET versions of Visual Basic, Visual C++, Visual J#, or any of more than 20 other CTS-compliant languages. A single assembly may contain multiple modules written in different .NET languages, and the types can reference each other just as if they were written in the same language.

In addition to the run time services, the .NET Framework also includes an extensive library of over 4000 classes organized into namespaces that provide a wide variety of useful functionality for everything from file input and output to string manipulation to XML parsing, to Windows Forms controls.

4. Conclusion
This paper outlines the special nature of software projects together with the difficulties of software project duration estimation. Two large groups of duration estimation techniques are presented. Then we count the advantages of using C# language for reducing software project complexity and duration.

References